

递归从这里开始，并且在这里得到最终值：234

value: [1, 5, 233, 7]
index: [0, 1, 2, 3]

cache里面装的都是每次递归得到的返回值。避免重复计算

predict(0, 3, True) returns 234				
index	0	1	2	3
0	0	5	5	234
1	-1	0	233	233
2	-1	-1	0	233
3	-1	-1	-1	0

- 重点：
- 1.递归函数入口
 - 2.递归中的关系怎么写
 - 3.递归终止条件（base case）
 - 4.怎么存入cache

```
def PredictTheWinner(self, nums):  
    """  
    :type nums: List[int]  
    :rtype: bool  
    """  
    # 递归函数  
    def predict(start, end, turn):  
        if cache[start][end] == -1: # 查找cache，如果里面没有值，就开始递归，如果有就直接返回  
            if start == end: # base case: Player 1优先选择数字  
                r = nums[start] if turn else 0 # if turn is True: r = nums[start] else r = 0  
            else: # 递归  
                if turn: # 轮到player1: 选择头部或者尾部的值+之前得到的最大值  
                    r = max(nums[start] + predict(start + 1, end, False),  
                           nums[end] + predict(start, end - 1, False))  
                else: # 轮到player2: 选择之前得到的最小值  
                    r = min(predict(start + 1, end, True), predict(start, end - 1, True))  
            cache[start][end] = r # 存入cache  
        return cache[start][end]  
  
    n = len(nums)  
    cache = [[-1 for i in range(n)] for j in range(n)] # 初始化cache  
    return 2 * predict(0, len(nums) - 1, True) >= sum(nums) # 递归函数入口,返回时判断player1 > player2
```

Player 1

predict(0,3,True)

max(num[0] + predict(1, 3, False), num[3] + predict(0, 2, False))

Player 2

min(predict(2, 3, True), predict(1, 2 True))

min(predict(1, 2, True), predict(0, 1, True))

Player 1

max(num[2] + predict(3, 3, False), num[3] + predict(2, 2, False))

max(num[1] + predict(2, 2, False), num[2] + predict(1, 1, False))

max(num[1] + predict(2, 2, False), num[2] + predict(1, 1, False))

max(num[0] + predict(1, 1, False), num[1]+ predict(0, 0, False))

if start == end:
 if True:
 return num[start]
 else:
 return 0

Player 2

递归在这行得到初始值

0

0

0

0

0

0

0

0